Hi there!

I’m Michael Schramm

github.com/wodka
twitter.com/wodkamichi

born in Salzburg

studying in Vienna

php development since 2003

co-founder of mymarket.io
e-commerce api solution

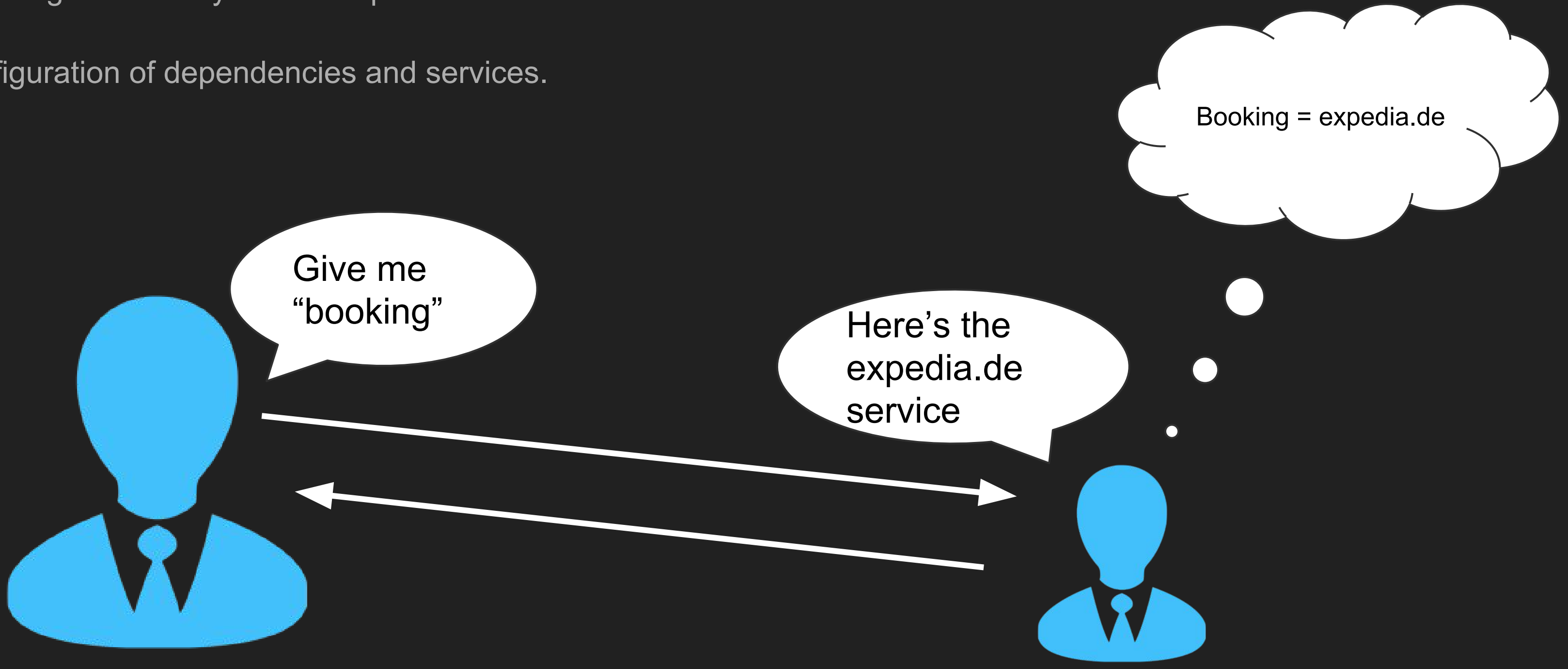
(Symfony2) Service Container

by Michael Schramm

What is a Service Container actually?

Object that manages the lifecycle and dependencies of services.

Requires configuration of dependencies and services.



Examples

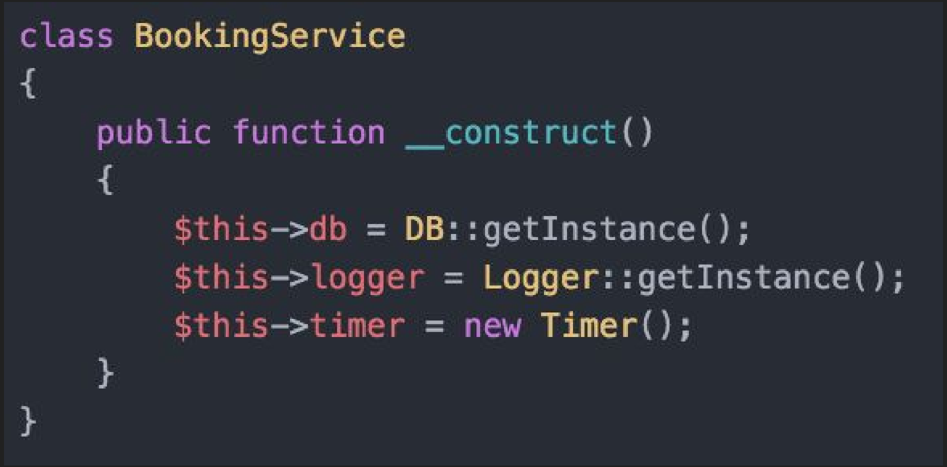
- JAVA: Spring Framework - Bean configuration
- PHP: Symfony2 Service Container
- PHP: Laravel Service Container
- PHP: Pimple
- JS: AngularJS
- …

you DI containers can be found almost everywhere!

Why bother with this at all?

- more configuration required
- there is already a working system
- why not hardcoding stuff?
  - faster
  - easy to understand

What could possibly go wrong here - right?

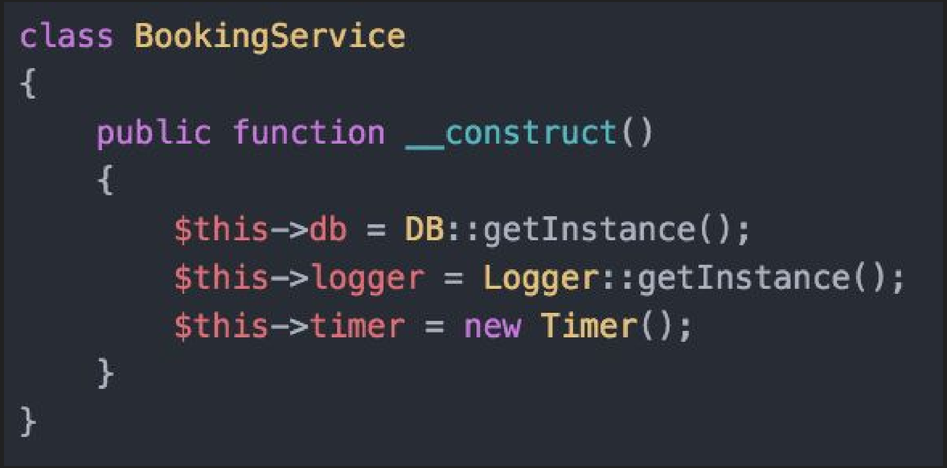


```php
class BookingService
{
    public function __construct()
    {
        $this->db = DB::getInstance();
        $this->logger = Logger::getInstance();
        $this->timer = new Timer();
    }
}
```

Why is this bad?

- Singletons = bad for testing - just try to mock the database or logger
- What if we have a second database with the booking service?
- …

All problems can be solved...
but there is a better solution!



```php
class BookingService
{
    public function __construct()
    {
        $this->db = DB::getInstance();
        $this->logger = Logger::getInstance();
        $this->timer = new Timer();
    }
}
```

Move the dependencies out!

- pass them in the construct or setter
- ideally use an interface

Doesn’t that look better?

```php
class BookingService
{
    public function __construct(Connection $db, LoggerInterface $logger)
    {
        $this->db = $db;
        $this->logger = $logger;
    }

    public function setTimer(TimerInterface $timer)
    {
        $this->timer = $timer;
    }
}
```

Till now this was applicable to all Service / DI Containers

Focus is now on Symfony2

Service Configuration

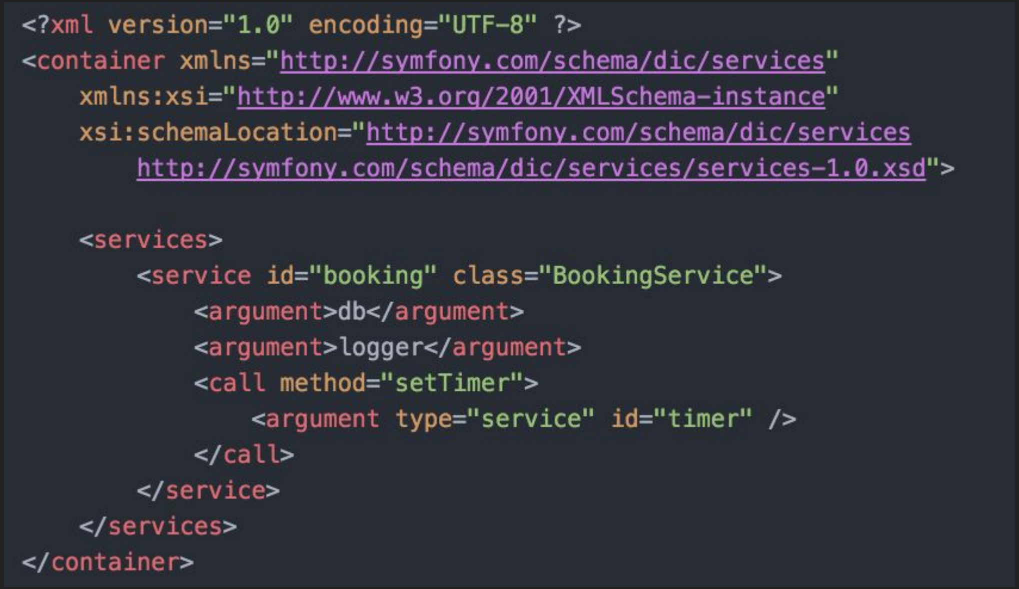
- XML
- YAML
- Annotations

Take a look at:

https://symfony.com/doc/current/book/service_container.html

XML Configuration

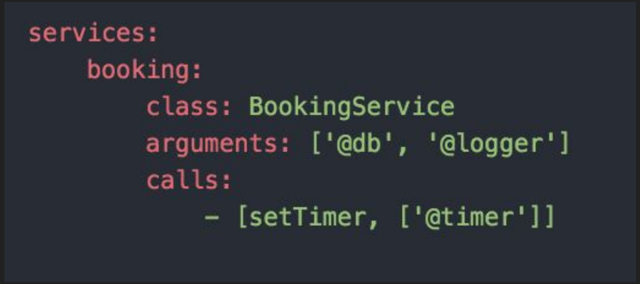
- most features
- bundle developers
- “bulky”



```xml
<?xml version="1.0" encoding="UTF-8" ?>
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
        http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <service id="booking" class="BookingService">
            <argument>db</argument>
            <argument>logger</argument>
            <call method="setTimer">
                <argument type="service" id="timer" />
            </call>
        </service>
    </services>
</container>
```

YAML Configuration

- almost identical to xml



```yaml
services:
    booking:
        class: BookingService
        arguments: ['@db', '@logger']
        calls:
            - [setTimer, ['@timer']]
```

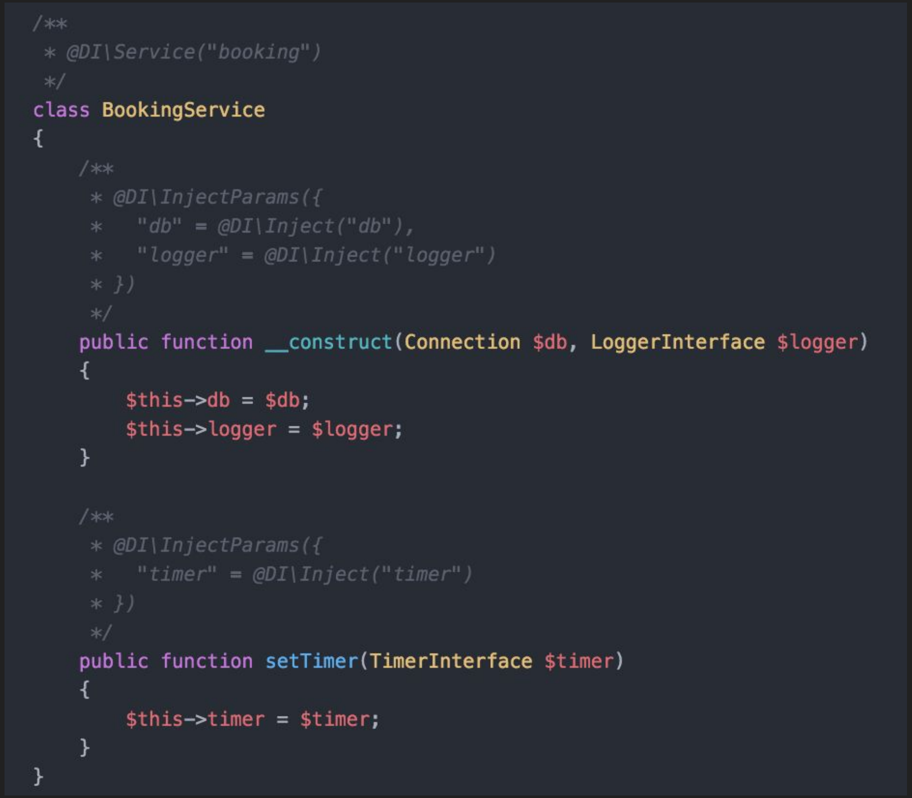
Annotation Configuration

- easiest to use
- right next to implementation
- nice for bundle extensions

http://jmsyst.com/bundles/JMSDiExtraBundle

Bundle Extensions:

- https://sonata-project.org/bundles/admin/3-x/doc/reference/annotations.html



```php
/**
 * @DI\Service("booking")
 */
class BookingService
{
    /**
     * @DI\InjectParams({
     *   "db" = @DI\Inject("db"),
     *   "logger" = @DI\Inject("logger")
     * })
     */
    public function __construct(Connection $db, LoggerInterface $logger)
    {
        $this->db = $db;
        $this->logger = $logger;
    }

    /**
     * @DI\InjectParams({
     *   "timer" = @DI\Inject("timer")
     * })
     */
    public function setTimer(TimerInterface $timer)
    {
        $this->timer = $timer;
    }
}
```

Advanced Concepts - tags

- linking on compile time
- whatever else :)

```
/**
 * @DI\Service("booking")
 */
class BookingService
{
    /* ... */

    public function addBookingProvider(BookingProviderInterface $pb)
    { /* add booking provider to list */ }
}

/**
 * @DI\Service()
 * @DI\Tag("booking.provider")
 */
class Expedia implements BookingProviderInterface
{}

/**
 * @DI\Service()
 * @DI\Tag("booking.provider")
 */
class Opodo implements BookingProviderInterface
{}
```

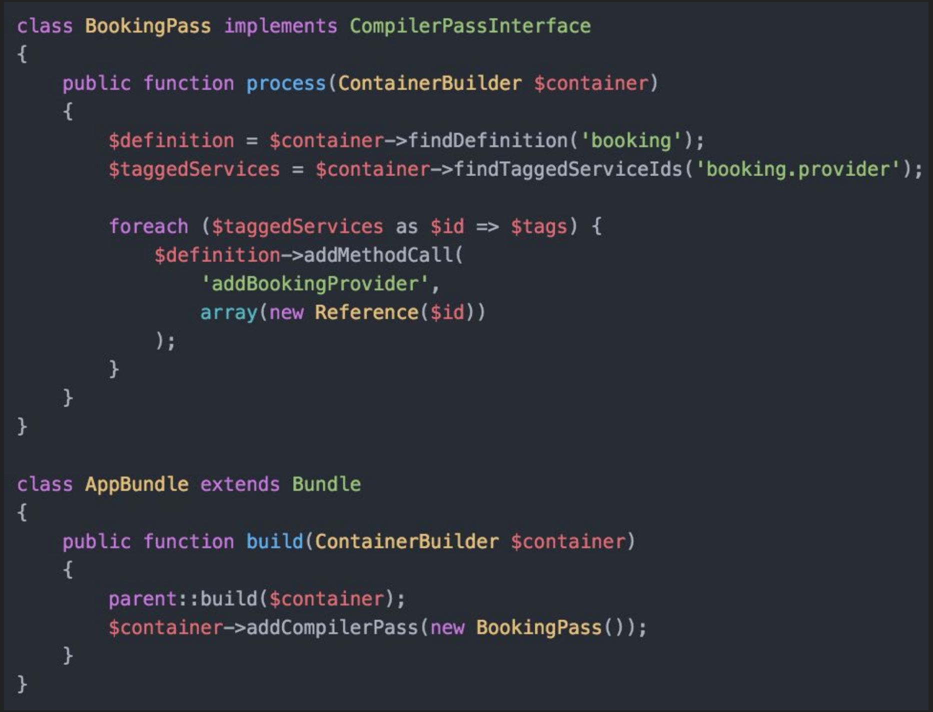
Advanced Concepts - tags

- linking on compile time
- whatever else :)

after compile both Opodo and Expedia will be available inside the bookingService

ATTENTION!

having many dependencies will take a toll on your application memory consumption



```php
class BookingPass implements CompilerPassInterface
{
    public function process(ContainerBuilder $container)
    {
        $definition = $container->findDefinition('booking');
        $taggedServices = $container->findTaggedServiceIds('booking.provider');

        foreach ($taggedServices as $id => $tags) {
            $definition->addMethodCall(
                'addBookingProvider',
                array(new Reference($id))
            );
        }
    }
}

class AppBundle extends Bundle
{
    public function build(ContainerBuilder $container)
    {
        parent::build($container);
        $container->addCompilerPass(new BookingPass());
    }
}
```

Advanced Concepts - proxy

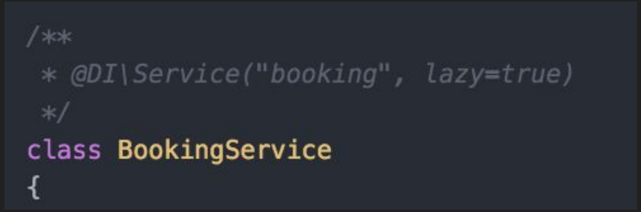
- only create instance if function is used
- limit circular dependencies
- Requirement for AOP

Check out: https://symfony.com/doc/current/components/dependency_injection/lazy_services.html

Setup is easy:

composer require ocramius/proxy-manager:~1.0

Now any service can be made “lazy”



```
/**
 * @DI\Service("booking", lazy=true)
 */
class BookingService
{
```

Advanced Concepts - AOP - pointcuts

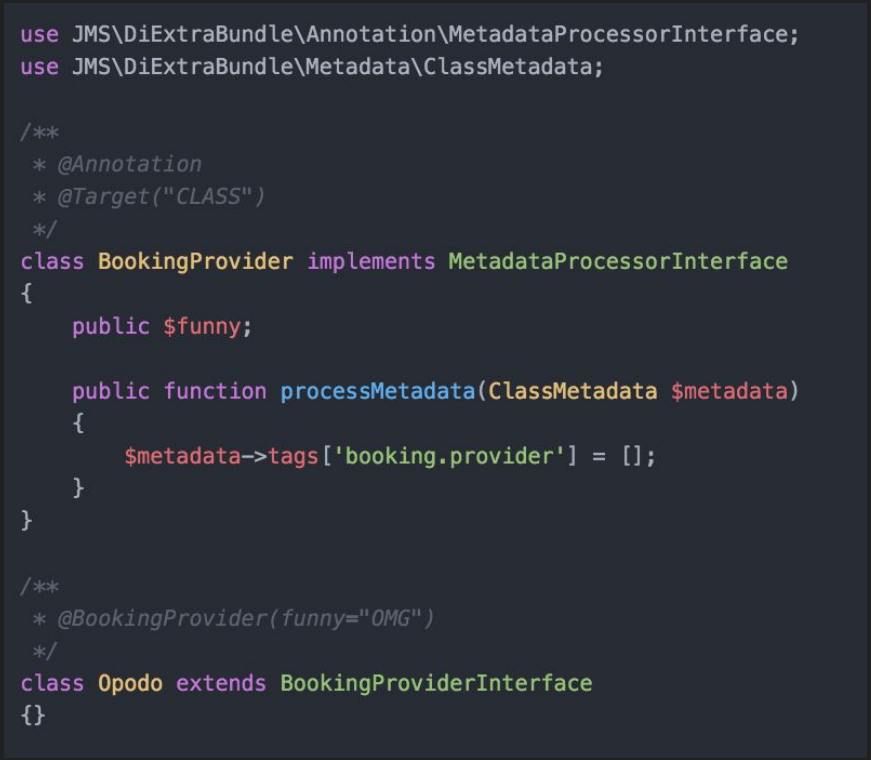
- regular expressions for function invocation
  - i.e. limit access for *Admin() functions to specific user group

```php
/**
 * @DI\Service
 * @DI\Tag("jms_aop.pointcut", attributes={"interceptor"="log.interceptor"})
 */
class LogPointcut implements PointcutInterface
{
    public function matchesClass(\ReflectionClass $class)
    {
        return true;
    }

    public function matchesMethod(\ReflectionMethod $method)
    {
        return preg_match('!admin|delete!i', $method->name);
    }
}
```

```php
/**
 * @DI\Service("log.interceptor")
 */
class LogInterceptor implements MethodInterceptorInterface
{
    /**
     * @DI\Inject("logger")
     */
    public $logger;

    public function intercept(MethodInvocation $invocation)
    {
        $this->logger->info(
            sprintf(
                'invoked method "%s".',
                $invocation->reflection->name
            )
        );

        // here we could block the execution
        return $invocation->proceed();
    }
}
```

Advanced Concepts - custom annotations

- great for bundle developers
  - SonataAdminBundle
- make code more readable



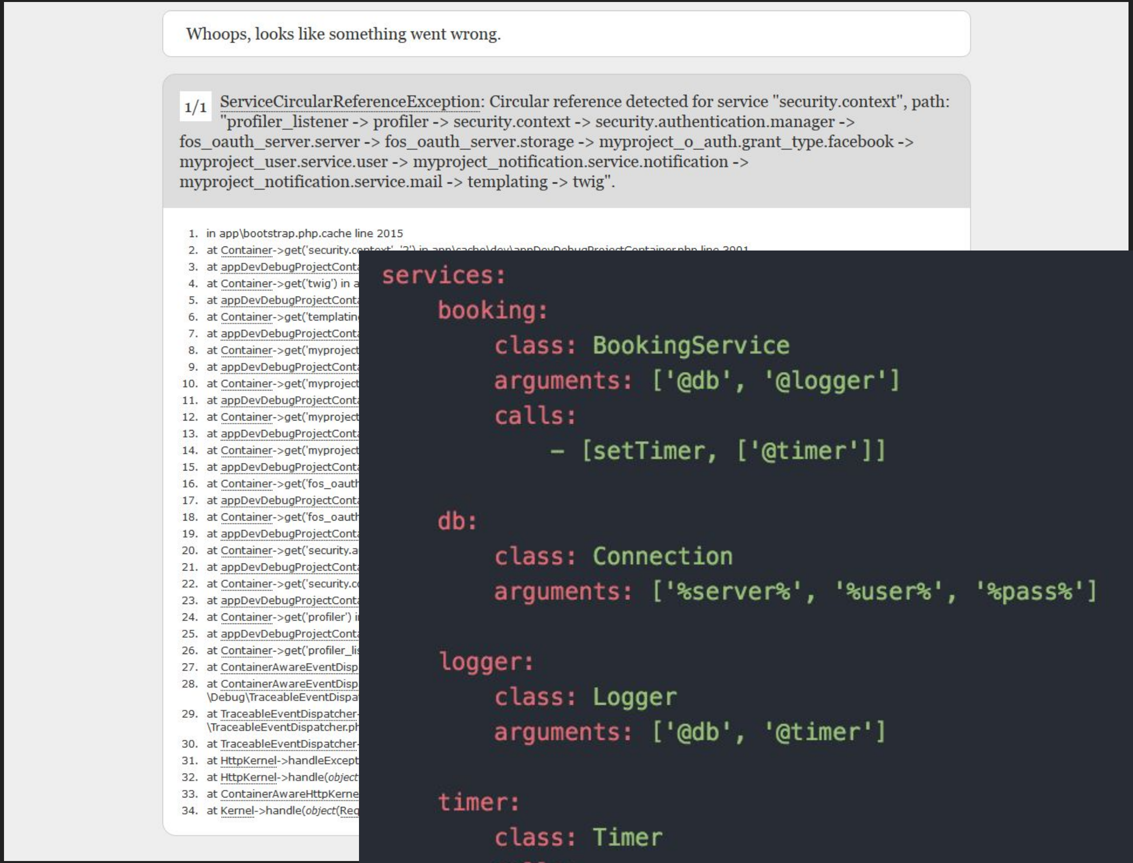
```php
use JMS\DiExtraBundle\Annotation\MetadataProcessorInterface;
use JMS\DiExtraBundle\Metadata\ClassMetadata;

/**
 * @Annotation
 * @Target("CLASS")
 */
class BookingProvider implements MetadataProcessorInterface
{
    public $funny;

    public function processMetadata(ClassMetadata $metadata)
    {
        $metadata->tags['booking.provider'] = [];
    }
}

/**
 * @BookingProvider(funny="OMG")
 */
class Opodo extends BookingProviderInterface
{}
```

Known errors - cycles

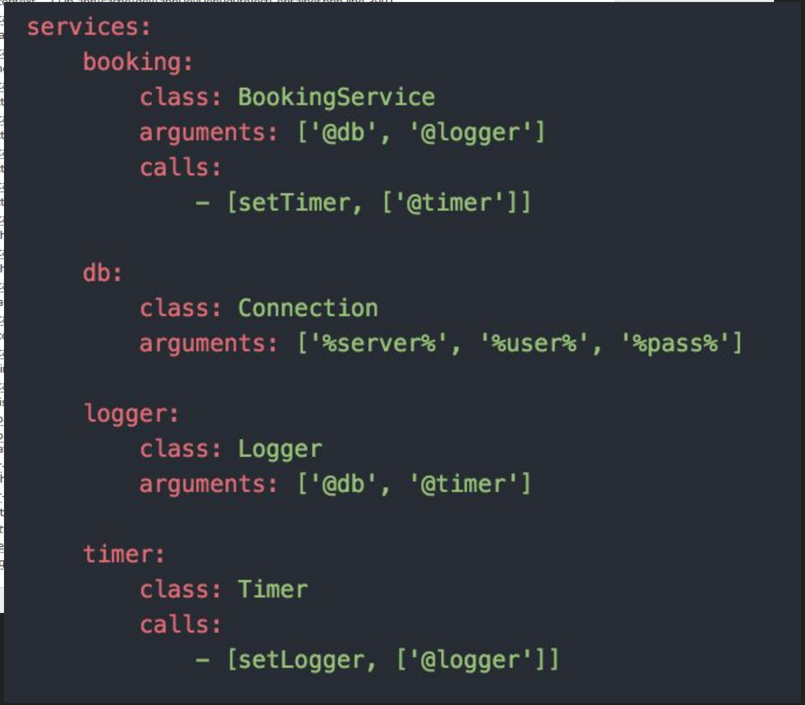
- happens quite easy
- hard to find
- proxies might help
- logger > timer > logger

What to check first:

- Twig extension dependencies
- Logger dependencies
- Any “general” dependency!



```
Whoops, looks like something went wrong.

1/1 ServiceCircularReferenceException: Circular reference detected for service "security.context", path:
"profiler_listener -> profiler -> security.context -> security.authentication.manager ->
fos_oauth_server.server -> fos_oauth_server.storage -> myproject_o_auth.grant_type.facebook ->
myproject_user.service.user -> myproject_notification.service.notification ->
myproject_notification.service.mail -> templating -> twig".

 1. in app\bootstrap.php.cache line 2015
 2. at Container->get('security.context', '2') in app\cache\dev\appDevDebugProjectContainer.php line 3901
 3. at appDevDebugProjectConta
 4. at Container->get('twig') in a
 5. at appDevDebugProjectConta
 6. at Container->get('templatin
 7. at appDevDebugProjectConta
 8. at Container->get('myproject
 9. at appDevDebugProjectConta
10. at Container->get('myproject
11. at appDevDebugProjectConta
12. at Container->get('myproject
13. at appDevDebugProjectConta
14. at Container->get('myproject
15. at appDevDebugProjectConta
16. at Container->get('fos_oauth
17. at appDevDebugProjectConta
18. at Container->get('fos_oauth
19. at appDevDebugProjectConta
20. at Container->get('security.a
21. at appDevDebugProjectConta
22. at Container->get('security.c
23. at appDevDebugProjectConta
24. at Container->get('profiler') i
25. at appDevDebugProjectConta
26. at Container->get('profiler_lis
27. at ContainerAwareEventDisp
28. at ContainerAwareEventDisp
    \Debug\TraceableEventDispa
29. at TraceableEventDispatcher-
    \TraceableEventDispatcher.ph
30. at TraceableEventDispatcher-
31. at HttpKernel->handleExcept
32. at HttpKernel->handle(object
33. at ContainerAwareHttpKerne
34. at Kernel->handle(object(Req
```



```yaml
services:
    booking:
        class: BookingService
        arguments: ['@db', '@logger']
        calls:
            - [setTimer, ['@timer']]

    db:
        class: Connection
        arguments: ['%server%', '%user%', '%pass%']

    logger:
        class: Logger
        arguments: ['@db', '@timer']

    timer:
        class: Timer
        calls:
            - [setLogger, ['@logger']]
```

questions?

Slides available on https://github.com/viennaphp